

# Qualitative Analysis of Integration Adapter Modeling

Daniel Ritter and Manuel Holzleitner

Technology Development, SAP SE  
Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany  
{daniel.ritter,manuel.holzleitner}@sap.com

**Abstract.** *Integration Adapters* are a fundamental part of an integration system, since they provide (business) applications access to its messaging channel. However, their modeling and configuration remain under-represented. In previous work, the integration control and data flow syntax and semantics have been expressed in the Business Process Model and Notation (BPMN) as a semantic model for message-based integration, while adapter and the related quality of service modeling were left for further studies.

In this work we specify common adapter capabilities and derive general modeling patterns, for which we define a compliant representation in BPMN. The patterns extend previous work by the adapter flow, evaluated syntactically and semantically for common adapter characteristics.

**Keywords:** Business Process Model and Notation (BPMN), Conceptual Modeling, Language Design, Message Endpoints, Quality of Service.

## 1 Introduction

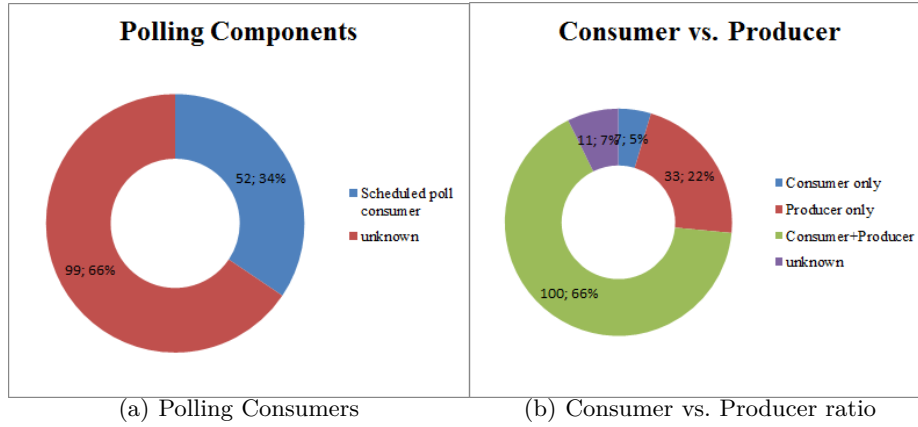
Although Enterprise Application Integration (EAI) continues to receive widespread focus by organizations offering them as means of integrating their conventional business applications with each other, with the growing amount of cloud applications and with their partners' systems, the integration adapter modeling is currently under-represented.

In this document we summarize a brief quantitative analysis of the currently used integration adapter types and their tasks according to the classification in [4]. The analysis is conducted on the widely used, open source integration system *Apache Camel* [2,1]. Based on the quantitative analysis, we conducted qualitative studies with integration experts in design thinking work shops and surveys. The studies target the modeling aspects of integration adapters from [4].

## 2 Apache Camel Component Analysis

To get a basic overview of existing integration components we implemented a system to introspect all Apache Camel [2,1] component bundles in the `org.apache.camel`

group in version 2.13, that were registered in the *Central Maven Repository* as of September 2014. With the help of byte code analysis, by using the open-source library *ASM*<sup>1</sup>, we automatically extracted basic capabilities from all of the 151 found adapters (cf. Listing 1.1). As such, we checked whether the consumer of an adapter extends from an scheduled poll consumer class to find out components which do not provide event-based consumers (cf. Listing 1.2, Figure 1(a)). We categorized the components whether they support producers, consumers or both (cf. Listings 1.3, 1.4, 1.5, Figure 1(b)), by checking if the component provides the necessary implementation classes for the *Camel Producer / Consumer* interface. Please note, that some components provide producer implementations that in fact consume messages rather than send messages, e. g., the *pop3* component can only be used to consume / poll for e-mails, although it provides an implementation for the producer interface. Such producer implementations are commonly used to compute a poll for messages, triggered by an event-message (event-based polling). This sub-categorization and the discovery of some, other capabilities could not be extracted automatically via byte code analysis, because this would require quite complex data-flow analysis, and remains to be done in the future.



**Fig. 1.** Message Endpoint Analysis of 119 Apache Camel Component bundles with in total 151 Components. The term “Component” is used equivalent to “Adapter”.

**Listing 1.1.** All Components (151)

```
ahc, ahc-ws, ahc-wss, amqp, apns, atmosphere-websocket, atom, avro, aws-cw,
aws-ddb, aws-s3, aws-sdb, aws-ses, aws-sns, aws-sqs, aws-swf, bean-validator,
box, cache, cmis, cometd, cometds, context, couchdb, crypto, cxf, cxfbean,
cxfrs, disruptor, disruptor-vm, dns, dropbox, ejb, elasticsearch, exec,
facebook, flatpack, fop, freemarker, ftp, ftps, gauth, geocoder, ghttp,
ghttps, glogin, gmail, google-drive, gora, gtask, guava-eventbus, hazelcast,
```

<sup>1</sup> OW2 Consortium, visited 02/2015: <http://asm.ow2.org/>

```
hbase, hdfs, hdfs2, http, http4, https, https4, ibatis, imap, imaps,
infinispan, irc, ircs, javaspaces, jclouds, jcr, jdbc, jetty, jgroups, jms,
jmx, jpa, jt400, kafka, kestrel, krati, ldap, lpr, lucene, metrics, mina,
mina2, mongodb, mqtt, msv, mustache, mvel, mybatis, nagios, netty, netty-http,
netty4, netty4-http, nntp, openshift, optaplanner, pop3, pop3s, quartz,
quartz2, quickfix, rabbitmq, restlet, rmi, rnc, rng, routebox, rss,
salesforce, sap-netweaver, schematron, scp, servlet, sftp, sip, sjms, smpp,
smpps, smtp, smtps, snmp, solr, solrCloud, solrs, somp, spark-rest, splunk,
spring-batch, spring-event, spring-integration, spring-ldap, spring-redis,
spring-ws, sql, ssh, stax, stream, string-template, twitter, velocity, vertx,
weather, websocket, xmlrpc, xmlsecurity, xmpp, xquery, yammer, zookeeper
```

**Listing 1.2.** Scheduled Poll Consumers components (52)

```
apns, atom, aws-s3, aws-sqs, bean-validator, cmis, cxfbean, dropbox, ejb,
facebook, freemarker, ftp, ftps, gora, hbase, hdfs, hdfs2, http, http4, https,
https4, ibatis, imap, imaps, jclouds, jpa, jt400, krati, msv, mustache,
mvel, mybatis, nntp, openshift, optaplanner, pop3, pop3s, rnc, rng, sftp,
smtp, smtps, splunk, sql, ssh, stax, string-template, twitter, velocity,
weather, xquery, yammer
```

**Listing 1.3.** Consumer only components (10)

```
atom, hazelcast, jetty, jmx, quartz, quartz2, rss, servlet, snmp, spark-rest
```

**Listing 1.4.** Producer only components (33)

```
ahc, aws-cw, aws-ddb, aws-sdb, aws-ses, aws-sns, crypto, dns, elasticsearch,
exec, fop, gauth, geocoder, glogin, gmail, jclouds, jdbc, jt400, ldap, lpr,
lucene, metrics, nagios, sap-netweaver, schematron, scp, solr, solrCloud,
solrs, spring-batch, spring-ldap, xmlrpc, xmlsecurity
```

**Listing 1.5.** Consumer & Producer components (108)

```
ahc-ws, ahc-wss, amqp, apns, atmosphere-websocket, avro, aws-s3, aws-sqs, aws-
swf, bean-validator, box, cache, cmis, cometd, cometds, context, couchdb,
cxf, cxfbean, cxfrs, disruptor, disruptor-vm, dropbox, ejb, facebook,
flatpack, freemarker, ftp, ftps, ghttp, ghttps, google-drive, gora, gtask,
guava-eventbus, hbase, hdfs, hdfs2, http, http4, https, https4, ibatis, imap,
imaps, infinispan, irc, ircs, javaspaces, jcr, jgroups, jms, jpa, kafka,
kestrel, krati, mina, mina2, mongodb, mqtt, msv, mustache, mvel, mybatis,
netty, netty-http, netty4, netty4-http, nntp, openshift, optaplanner, pop3,
pop3s, quickfix, rabbitmq, restlet, rmi, rnc, rng, routebox, salesforce, sftp,
sip, sjms, smpp, smpps, smtp, smtps, splunk, spring-event, spring-
integration, spring-redis, spring-ws, sql, ssh, stax, stomp, stream, string-
template, twitter, velocity, vertx, weather, websocket, xmpp, xquery, yammer,
zookeeper
```

### 3 Qualitative Analysis of the Adapter Modeling Approach

The adapter characteristics and the modeling approach were part of a survey-based analysis with 20 integration experts. The surveys were partially conducted as interviews (due to expert availability). The results are briefly discussed subsequently and have been anonymized, where necessary.

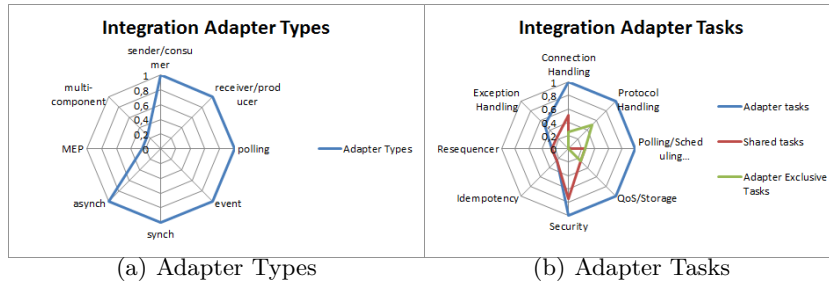
The survey was set up mostly with free-text fields and multiple-choice sections, the questions and the diagrams from the paper [4]. Not all participants answered all questions.

### 3.1 Adapter Characteristics

Some of the adapter characteristics have been already analyzed experimentally in Section 2. However, we cross-checked some aspects (i. e., adapter types and tasks) in the survey as introductory questions to gain some more information on the participant's background. Figure 2(a) shows that all participants know the basic adapter types and their separation into polling or event-based consumers and producers, as well as synchronous (synch) and asynchronous (asynch) communication. Surprisingly, not all survey participants differentiate between synch/asynch communication and the *Message Exchange Patterns* (MEPs) `inOnly` / `inOut` (cf. [4]). Same applies to the Apache Camel multi-component modeling, which allows to specify multiple transport protocols in one Camel component.

For the adapter tasks, all participants named (technical) connection and protocol handling, scheduling, storage and Quality of Service (QoS) support as well as transport-level security aspects like encryption, authentication. Figure 2(b) shows the summarized responses. Notably, most of the participants did not name the resequencer and idempotency characteristic separately, but saw them included in the QoS support. Others correctly named idempotency as task of the receiver, which is considered the only save case. However, if a receiving backend system does not support that, the integration system (e. g., producer adapter) has to take over. The exception handling was mentioned by little less than half of the participants, which might show a lack of awareness for the topic and could require an additional educational offering.

The exception handling, resequencer, and security (i. e., message-level: encrypt, signing, not communication channel security) were mostly seen as shared tasks between the adapters and integration pipelines. Topics that were seen exclusively important for the adapter are protocol / format handling.



**Fig. 2.** Adapter Types and Tasks (100% equals to 1.0)

### 3.2 Adapter Flow

The *Adapter Flow* (AF) was considered a requirement for many scenarios by all participants. However, most participants argued for consumer AFs, due to the possibility of message pre-processing before handing the message to the integration process. Apart from some exceptions, the post-processing could be moved to the integration process. Prominent exceptions were seen in the idempotent message handling and potential flexibility to build scenario-specific extensions. The modularity and architectural aspects were only brought up by few participants. For them it is important to be able to run the AFs on non-integration runtime systems (e.g., Extraxt/Transform/Load or Complex Event Processing-Tools for more data-centric processing).

### 3.3 Bridges

Most participants like the explicit modeling of “bridges”, however, have doubts about complexity: nobody wants to model these constructs piece by piece. Hence the experts proposed a pattern-based modeling approach similar to the *Enterprise Integration Pattern* (EIP) modeling [3], which could allow to drag&drop the bridges from the palette. One problem that was mentioned is about multiple the placement of the same pattern and simultaneous changes afterwards. For that, the design time tool would have to offer refactorings for multiple patterns, at the same time, through “where-used” support, and capabilities that allow to capture an adapted pattern as user-defined, “re-usable pattern”.

The participants require such a modeling approach for scenario-based variations of the default bridge implementations. For simple adapters (e.g., that do not require special pre- / post-processing or QoS support), the participants mostly argued for the BPMN *Message Flow*-based modeling, which means “over-defining” the BPMN constructs and adding a property sheet.

### 3.4 Security Aspects

The explicit modeling of security relevant information like certificates/key stores, was controversially discussed. Partners and consultants require more configuration capabilities for certificate handling and an explicit association from the AFs. That means, a key store could be referenced from multiple integration processes.

On the other hand, more business-near experts argues that the explicit modeling will be too complex in most cases.

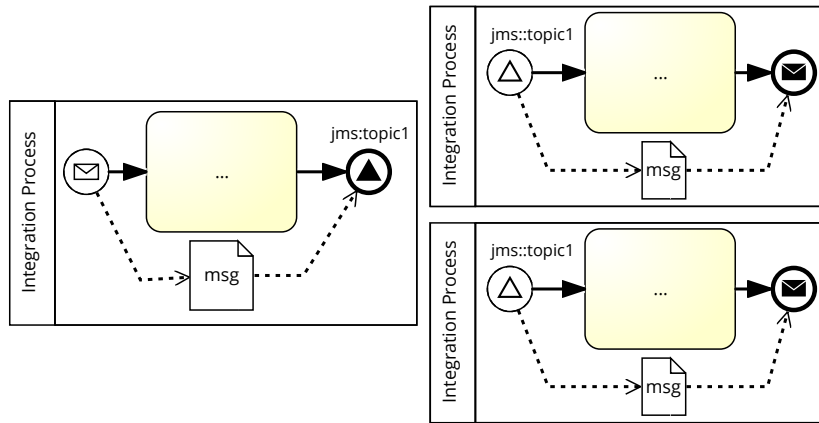
### 3.5 Message Queuing

For message queuing the participants stated that the diagrams are understandable as long as only two integration processes are connected via queues in one IFlow. Variants in which four integration processes were connected to one single messaging system or duplicated messaging systems in the same IFlow were used, were regarded as too complex. The transactional dequeuing variant was received

well als it makes the transactional boundaries visible, however the modeling of the boundaries might not be required in all cases as they might be derived from the tasks themselves.

As an alternative, the messaging system modeled as a BPMN participant allows to show the inner workings of the messaging system, which was received as informative but too complex for some users to understand. On the one hand, technical and implementation-oriented users see the need for a flexible and comprehensive modeling of the queuing and the messaging system. On the other hand, business-oriented users would like to have an high-level overview on the overall integration by hiding technical details such as messaging and queueing in general (only noticeable as QoS).

All participants agreed that the modeling approach via a BPMN data store is superior to the alternative modeling approach via BPMN signals (cf. Topic Modeling in Figure 3). Finding the matching names for the topic on one diagram without a visual hint via connectors reduces the overall understandability of an IFlow by making connectivity too implicit.



**Fig. 3.** Alternative Topic Modeling using *BPMN Signal End Event*.

### 3.6 Reply Flow

The reply flow was received well in the survey as it allows to model important post-processing steps in the IFlow. Also other use-cases were identified where the modeling of the reply flow is required. However, here the reply flow adds complexity to the overall IFlow, thus identifying the need for good tool support that allows for hiding / un-hiding the reply flow.

### 3.7 Cross-Tenant, Network Modeling

For cross-tenant connectivity between IFlows the modeling via enqueue-/dequeuing on a messaging system it gets visible that they are communicating in an asynchronous fashion. Thus, the messaging system modeled as BPMN data store can function as the boundary to other IFlows (in other tenants) and can be used to split multiple diagrams to reduce the complexity. However, to get an high-level overview it must be possible to combine all connected diagrams to an high-level overview.

### 3.8 Quality of Service Modeling

In the survey some participants prefer to model QoS on sender adapter as a property without explicitly model redelivery, resequencing and de-duplication via idempotency repository. Other participants prefer the improved capabilities and flexibility of an integration system supporting detailed QoS modeling. However, it is important to note that some of the QoS tasks are preferably located either on the receiver or the sender adapter, e. g., the redelivery should usually be done on sender side and the deduplication should be modeled in the receiver adapter.

## 4 Conclusions

The work relates to the classification of integration adapters and modeling approaches described in [4]. Based on a quantitative analysis (i. e., , real-world case studies, integration expert interviews and surveys), this work gives a brief, but comprehensive overview of integration adapter types and tasks. With the qualitative user studies modeling alternatives are evaluated and user preferences captured (e. g., the value of a modular, pattern-based and explicit modeling approach) as well as its downsides (e. g., modeling complexity, technical diagrams). These identified aspects will be further analyzed in quantitative user studies and an extension of the modeling language (e. g., conditional data flows).

## Acknowledgments.

We thank all participants of the expert workshops, surveys and interviews as well as Volker Stiehl and Ivana Trickovic for valuable discussions on BPMN.

## References

1. J. Anstey and H. Zbarcea. *Camel in Action*. Manning, 2011.
2. ApacheFoundation. Apache camel. <http://camel.apache.org/>, 2015.
3. D. Ritter. Using the business process model and notation for modeling enterprise integration patterns. *CoRR*, abs/1403.4053, 2014.
4. D. Ritter and M. Holzleitner. Integration adapter modeling. In *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015 (accepted), Stockholm, Sweden, June 8-12, 2015. Proceedings*, 2015.